Theoretical Computer Science 457 (2012) 27–34

Contents lists available at SciVerse ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

On suffix extensions in suffix trees☆

Dany Breslauer[a], Giuseppe F. Italiano[b,*]

[a] Caesarea Rothschild Institute for Interdisciplinary Applications of Computer Science, University of Haifa, Haifa, Israel
[b] Dipartimento di Informatica, Sistemi e Produzione, Università di Roma "Tor Vergata", Rome, Italy

ARTICLE INFO

Article history:
Received 8 September 2011
Received in revised form 9 July 2012
Accepted 18 July 2012
Communicated by M. Crochemore

Keywords:
Suffix trees
String algorithms

ABSTRACT

Suffix trees are inherently asymmetric: prefix extensions only cause a few updates, while suffix extensions affect all suffixes causing a wave of updates. In his elegant linear-time on-line suffix tree algorithm Ukkonen relaxed the prevailing suffix tree representation and introduced two changes to avoid repeated structural updates and circumvent the inherent complexity of suffix extensions: (1) open ended edges that enjoy gratuitous leaf updates, and (2) the omission of implicit nodes.

In this paper we study the implicit nodes as the suffix tree evolves. We partition the suffix tree's edges into collections of similar edges called *bands*, where implicit nodes exhibit identical behavior, and generalize the notion of open ended edges to allow implicit nodes to "float" within bands, only requiring updates when moving from one band to the next, adding up to only $O(n)$ updates. We also show that internal implicit nodes are separated from each other by explicit suffix tree nodes and that all external implicit nodes are related to the same periodicity. These new properties may be used to keep track of the waves of implicit node updates and to build the suffix tree on-line in amortized linear time, providing access to all the implicit nodes in worst-case constant time.

1. Introduction

Suffix trees and *suffix arrays* are perhaps the most prevalent data structures in the study of string algorithms, with numerous applications [2,14,22,29]. Weiner [38] introduced suffix trees and gave a reverse right-to-left on-line algorithm for their construction in linear-time. McCreight [31] gave a linear-time left-to-right algorithm that is not on-line since it must process sufficient "lookahead" of text symbols to find the correct insertion points in the suffix tree. Suffix arrays, that were introduced by Manber and Myers [30], provide similar theoretical benefits to suffix trees, but are much more efficient in practice thanks to their use of efficient array representation, avoiding the extra space and irregular memory access patterns inherent in pointer based data structures. The compact and efficient suffix array representation in a contiguous memory block, however, is less amenable to *on-line* construction, because the eventual insertions in the middle of an array would be costly. In fact, the fastest existing linear-time suffix array construction algorithms over *large integer alphabets* [24–26] and their earlier linear-time suffix tree counterparts [17,18] usually use bucket sort and other techniques that are unfortunately *off-line*.

Ukkonen's [37] on-line algorithm is often regarded as the simplest and the most intuitive among the suffix tree construction algorithms. To develop an efficient linear-time "on-line" left-to-right suffix tree algorithm, Ukkonen had to

☆ A preliminary version of this paper was presented at SPIRE 2011 — the 18th International Symposium on String Processing and Information Retrieval (Breslauer and Italiano, 2011 [10]).

* Corresponding author. Tel.: +39 06 7259 7394; fax: +39 06 7259 7460.
E-mail address: italiano@disp.uniroma2.it (G.F. Italiano).

0304-3975/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2012.07.018

relax the prevailing representation of suffix trees in order to avoid repeated structural updates to the suffix tree. In particular, Ukkonen introduced two changes: (1) *open ended edges* leading to suffix tree leaves that represent ever growing suffixes of the text, introducing gratuitous leaf updates; and (2) the omission of *implicit non-branching suffix nodes* until such nodes can be made explicit by inserting branching nodes and leaves. Analogous issues arise in an incrementally maintained suffix array or the closely related Burrows–Wheeler Transform compressed text [33,34].

While open ended edges are an elegant concept that was celebrated by Geigerich and Kurtz [21] who speculate that "if Weiner had seen this idea in 1973, he would have designed Ukkonen's algorithm then", the omission of "implicit" internal suffix nodes is necessitated by their frequent updates. Gusfield [22] calls Ukkonen's intermediate suffix trees "implicit suffix trees" and Ukkonen [37] writes that "when explicit final states are needed in some application, they are obtained gratuitously by adding to $\mathcal{T}$ an end marking symbol that does not occur elsewhere in $\mathcal{T}$". However, it might be costly to follow this suggestion repeatedly. Indeed, always updating explicitly all the implicit nodes in a text of length n takes at least $\Omega(n \log_{|\Sigma|} n)$ updates and up to $O(n^2)$ updates, depending on the structure of the text (e.g. $a^n b a^n c$). The need for an end marking symbol to complete the construction is related to subsequential transducers [5,13] where the computation is completed only upon a terminating input symbol.

Throughout the paper, we assume that the input alphabet Σ has constant size and refer to edges leading to suffix tree leaves either as open ended edges or as external leaf edges and to the other edges as internal edges. Our contributions in this paper are two-fold. First, we use properties of periodicities in the text to draw conclusions about the locations of the implicit nodes as the suffix tree evolves. Specifically, we prove that:

1. there can be only one implicit node floating within each internal suffix tree edge, and that,
2. external open ended suffix tree edges may contain several implicit floating nodes, but all such implicit nodes are related to the same periodicity; these external implicit nodes do not require updates until they branch out since they never reach the end of an open ended edge.

These two properties stem from the fact that longer irregular periods which are not multiples of a string's shortest period correspond to terminated prefix periods that introduce branching nodes in the suffix tree, separating the implicit nodes. While periodicity properties are often used in comparison-based string matching algorithms, suffix trees capture all the internal structure of the text and algorithms typically do not need periodicity properties in their construction. Periodicity properties were occasionally used either implicitly or explicitly, however, in several suffix tree based algorithms [3,12,23,28].

We then push Ukkonen's approach one step further and generalize the notion of open ended edges to allow implicit nodes to "float" within suffix tree edges, requiring updates only when an implicit node moves from one edge to the next. We group the suffix tree edges containing implicit floating nodes into *bands*, which are collections of similar edges whose *both* endpoints are connected by suffix links, where implicit floating nodes exhibit identical behavior. We prove that if we maintain only one *implicit node representative* in each band, then the number of representative updates throughout the suffix tree construction drops to $O(n)$, significantly fewer than the $O(n^2)$ total explicit updates of all the implicit nodes.

Based on these properties, we present a linear-time on-line algorithm that maintains the representatives of the implicit nodes in each band via an auxiliary data structure, providing access to all the implicit nodes in worst-case constant time via queries that produce the implicit nodes on any given suffix tree edge, enabling algorithms that might require these nodes to use the intermediate implicit suffix trees without continuously adding the end marking symbol.

The remainder of the paper is organized as follows. We review suffix trees in Section 2 and Ukkonen's on-line algorithm in Section 3. We then characterize the locations of the implicit nodes in Section 4 and show how to maintain the implicit nodes in Section 5. We conclude with some final remarks and open problems in Section 6.

2. Suffix trees

The PATRICIA tree, introduced by Morrison [32], is an extremely useful trie data structure [20] representing strings over an alphabet Σ. In the trie, each tree node may have up to $|\Sigma|$ outgoing edges, where each edge is labeled with a distinct alphabet symbol. The node label is defined as the concatenation of the edge labels on the unique path from the root to the node. PATRICIA trees can grow quite large, however, because there might be long non-branching paths even when used to represent the set of suffixes of one given text string. To overcome this problem, Weiner [38] introduced *position trees*, which are more commonly called *suffix trees*. In suffix trees long paths of nodes with one child are compacted into one single edge: to save space, the edge labels with the concatenated symbols are represented by their starting and ending positions in the reference text, and thus each edge takes only constant space.

Given a text w, the *suffix tree* of w is a rooted tree with edges and nodes that are labeled with substrings of w. The suffix tree satisfies the following properties:

1. edges leaving any given node are labeled with non-empty strings v that start with different alphabet symbols (v is a substring of w);
2. each node is labeled with a string v formed by the concatenation of the edge labels on the path from the root to that node (v is a substring of w);

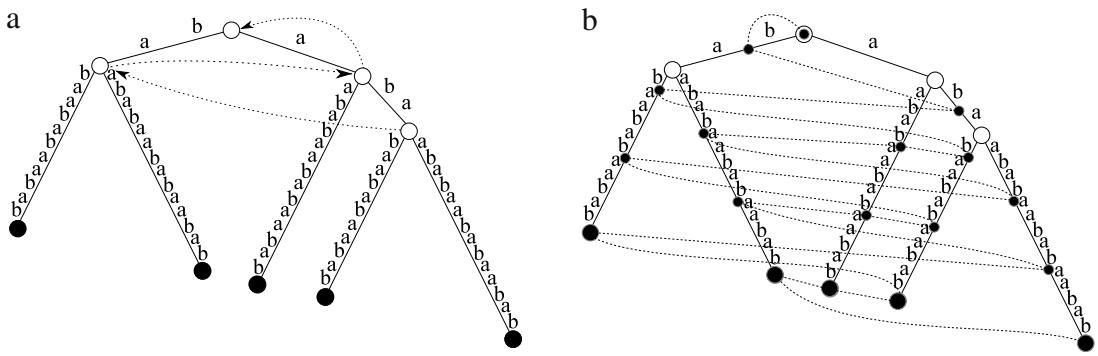

Fig. 1. (a) The suffix tree for the text "*abaababaabaabab*" with internal node suffix links. (b) The suffix tree with implicit nodes and the suffix chain. The suffix link trie consists of the suffix chain in figure (b) plus the suffix links shown in (a).

3. each branching internal (non-leaf) node has at least two descendants (the root may be an exception in the degenerate case when a string is empty or it is formed by repetitions of a single alphabet symbol);
4. for each substring v of w, there exists a vertex labeled u, such that v is a prefix of u.

It is a common practice to append at the end of the text w a special unique alphabet symbol \$, which does not appear anywhere within w. This guarantees that the suffix tree has exactly $|w| + 1$ leaves that are labeled with all the distinct non-empty suffixes of w\$. The number of branching internal nodes is no larger than $|w|$. However, in on-line algorithms that construct the suffix trees for a left-to-right streaming text, it is not possible to append the special alphabet symbol \$ at each step. Therefore, an on-line algorithm must deal also with suffix tree nodes representing text suffixes which may not be branching out of the tree. Such text suffixes might end at internal branching suffix tree nodes, but also in the middle of suffix tree edges. We use the convenient notation of associating an edge between parent u to child v with the child suffix tree node v. One can realize such suffix trees by taking the suffix trees with the special terminating symbol \$, and removing all edges that are labeled only with the symbol \$ and the leaves connected to these edges. This process may introduce internal non-branching nodes into the suffix tree, which are sometimes called *implicit nodes*, that may or may not be represented explicitly in the suffix tree. Such suffix trees were called *extended suffix trees* by Breslauer and Hariharan [8] and *implicit suffix trees* by Gusfield [22].

Weiner [38], McCreight [31] and Ukkonen [37] all augment the suffix trees with shortcuts called *suffix links*, that are used to efficiently traverse the suffix tree. We define the *suffix link* for a suffix tree node labeled $v = au, a \in \Sigma$, to be a pointer to the suffix tree node labeled with its suffix u, obtained by chopping off v's first symbol a. Suffix links are always defined for each internal branching node. If the node $v = au$ branches with edges that begin with alphabet symbols b and c, $b \neq c$, then the suffix tree also contains the substrings ub and uc and there must be a node labeled u, branching on the symbols b and c. The situation with suffix tree leaves is a little more complicated. Leaves clearly represent text suffixes, since only suffixes of the text end abruptly on their right side. If the text is terminated with a unique symbol, then all suffixes of the text are leaves and, therefore, if $v = au$ is a leaf then its suffix u must also be a leaf. However, if the text is not terminated, then the suffix u might be an implicit non-branching node in the middle of an edge or coincide with an existing branching node.

Since each non-root node has one suffix link and suffix links cannot introduce cycles, suffix links define a tree rooted at the suffix tree root. In fact, if each tree edge between nodes $v = au$ and u is labeled with the alphabet symbol a, this tree is actually an un-compacted trie, which is a subtree of the suffix trie for the reversed text. In this paper we maintain the edges of this trie in both directions (McCreight and Ukkonen only need edges pointing towards the root), and call this tree the *suffix link trie*. The path in the suffix link trie from the longest leaf representing the full text to the root goes through all the suffixes of the text, which are the only substrings that get extended while the input text is processed from left to right. We call this path the *suffix chain* (see Fig. 1).

Lemma 2.1. *The suffix chain can always be partitioned into the following consecutive segments:* (1) *leaves;* (2) *external implicit nodes within leaf edges;* (3) *internal implicit nodes within internal edges; and* (4) *implicit nodes that coincide with explicit nodes.*

Proof. Let v be a suffix of the text. If v is not a leaf in the suffix tree, then it can be extended and so can all its suffixes, making all suffixes of v implicit nodes. If v is not an external implicit node, then it can be extended to some branching node and so can all its suffixes, making all suffixes of v internal implicit nodes. Similarly, if v coincides with some branching node, then all its suffixes must also coincide with branching nodes. □

3. Ukkonen's algorithm

Ukkonen [37] noted the great resemblance between his and McCreight's algorithms, by observing that "in its final form our algorithm is a rather close relative of McCreight's method". The parallels between McCreight's, Ukkonen's and Weiner's algorithms have been studied by Geigerich and Kurtz [21], who showed that the difference between Ukkonen's and

McCreight's algorithms boils down to their "control structure" and that, essentially, they update the suffix tree in exactly the same insertion batches. McCreight's algorithm reads ahead sufficient "lookahead" symbols to find the final insertion points of new suffix leaves, while Ukkonen's algorithm holds off inserting new suffixes until sufficient text symbols have been scanned to insert new branching suffix leaves, omitting the implicit nodes.

While the input text is streamed on-line from left to right, all the suffixes of the text are extended with each right text extension. Ukkonen observed that once some text suffix is a leaf, it will remain forever a leaf after all future extensions. By labeling the external suffix tree edges leading to leaves "open ended", reaching to the current *growing* end of the text, Ukkonen invented an automatic gratuitous extension mechanism for these edge labels. Unfortunately, the remaining text suffixes that are not represented by leaves still move around the suffix tree, and it would be too costly to update all their locations. Hence, Ukkonen chose not to update these "implicit" nodes explicitly.

Ukkonen's on-line algorithm maintains the *active suffix*, the longest suffix of the text that has not branched out to become a leaf, which is the *longest repeated suffix* of the text that appeared previously in the text. By Lemma 2.1, any suffix of the active suffix must have also appeared previously and has not branched out yet. If upon the next symbol the active suffix cannot be extended within the suffix tree, an *insertion batch* creates leaves for all suffixes between the old active suffix and up to the new active suffix. In particular:

1. those implicit nodes on the suffix chain that were in the middle of an edge must branch out by creating an internal branching node splitting the edge and inserting a leaf; and
2. those implicit nodes that coincided with an explicit node branch out by creating a leaf hanging off the existing explicit node.

Ukkonen's algorithm maintains the current active suffix by a pointer to a suffix tree node and an offset within a suffix tree edge that are updated while tracing the suffix tree, selecting the appropriate branch at each internal suffix tree node according to the first symbol on the branching edges. During an insertion batch, the algorithm follows the suffix links to find the next active suffix. While following suffix links, more suffix tree nodes may appear on the path between the suffix tree node and the offset representing the implicit node and the representation must be updated to the *canonical* representation specifying implicit nodes by their offset relative to the beginning of the edge where they are located. Ukkonen's algorithm, like McCreight's algorithm, only has to navigate the suffix tree by selecting edges at each branching node according to their first branching symbol, quickly moving down the suffix tree path towards the implicit node. The total amount of work is amortized to linear time.

4. The locations of implicit nodes

Periodicity is often used in efficient string matching algorithms. However, suffix trees and related index data structures that express all internal repetition structure of a string typically rely instead on the mechanics of maintaining various graph pointers and on identifying alphabet symbols via direct array access. In this section we use simple periodicity properties to sort through the locations of implicit nodes. Before doing that, we need to review some basic terminology.

A string u is *a period* of a string w if w is a prefix of u^k for some integer k, or equivalently if w is a prefix of uw. The shortest period of w is called *the period* of w and w is called *periodic* if it is at least twice as long as its period. If v is a prefix of w, then the period of v is said to *continue* in w if v and w have the same period and otherwise the period of v *terminates* in w. A string v is called a *border* of w, if v is both a prefix and a suffix of w. By these definitions v is a border of $w = uv = vu'$ if and only if u is a period of w, and therefore, u is the shortest period of w if and only if v is the longest proper border. The following *Periodicity Theorem* is due to Fine and Wilf [19].

Theorem 4.1. *If a string u has periods of length p and q, and its length $|u| \geq p + q - \gcd(p, q)$, then u also has a period of length $\gcd(p, q)$.*

The following simple observation connects periods and borders of strings to their suffix trees and suffix link tries.

Lemma 4.2. *A string v is an ancestor of w both in the suffix tree and in the suffix link trie if and only if v is a border of w.*

Proof. Recall that v is an ancestor of w in the suffix tree if and only if v is a prefix of w, i.e., $w = vu'$. Similarly, v is an ancestor of w in the suffix link trie if and only if v is a suffix of w, i.e., $w = uv$. Therefore, v is an ancestor of w both in the suffix tree and in the suffix link trie if and only if v is a border of w. □

We will next exploit the connection given in Lemma 4.2 to prove the following two properties on suffix trees: (1) there might be only one implicit node within each internal edge, and (2) although external leaf edges may contain many implicit nodes, those implicit nodes enjoy some nice structural properties.

Theorem 4.3. *An internal suffix tree edge may contain at most one implicit node. This node may coincide with the branching node at the end of the edge.*

Proof. Let w be an internal branching suffix tree node. Assume by contradiction that there exist two different implicit nodes u and v on the edge leading to w, such that $|u| < |v| \leq |w|$. Node u is clearly an ancestor of v in the suffix tree. Since implicit nodes always represent suffixes of the text, it is not difficult to see that u must be an ancestor of v also in the suffix link trie. Then, by Lemma 4.2, u is a border of v and $v = xu$ has a period x. Let vy be the longest prefix of w which continues that

period x of v. If vy is a proper prefix of w, then let vya be the prefix of w where that period x terminated and let $vyb = xuyb$, which is not in the suffix tree, be a string that continues the periodicity, such that uyb is a prefix of $vy = xuy$. Otherwise, if $vy = w$, then w is an internal branching node and it must have at least two outgoing edges, at least one of which contains $wa = vya$ that does not continue that period x, while $wb = vyb = xuyb$, $a \neq b$, may continue the periodicity x. In either case, the suffix tree contains both uya and uyb, $a \neq b$, and uy must be a branching suffix tree node, either between u and v if $|u| \leq |uy| < |v|$ or between v and w if $|v| \leq |uy| < |w|$ contradicting the assumption that u, v and w are on the same edge. □

Leaf edges may each contain several external implicit nodes. The following theorem characterizes all the external implicit nodes and shows that these nodes must obey a simple arithmetic progression formula derived from the total number of external implicit nodes, the longest external implicit node and the period length of its associated leaf.

Theorem 4.4. *Let $v_0, \ldots, v_{k-1}$ be all the external implicit nodes ordered in decreasing length, and let $w_0, \ldots, w_{k-1}$ be the leaves at the end of their respective edges. Observe that multiple external implicit nodes might be on the same external edges leading to the same leaves. Let $p = |w_0| - |v_0|$, $m = \lfloor k/p \rfloor$ and $r = k - pm$. Then:*

(1) *The implicit node v_i has length $|v_i| = |v_0| - i$, $i = 0, \ldots, k-1$.*
(2) *There are at most p distinct leaves with lengths $|w_i| = |w_0| - i$, $i = 0, \ldots, \min\{p, k\} - 1$. For $i = p, \ldots, k-1$, we have $w_i = w_{i-p}$.*
(3) *The implicit nodes within the leaf edges to w_i have lengths*

$$|w_i| - \ell p \quad \begin{cases} \text{for } \ell = 1, \ldots, m+1, & \text{if } 0 \leq i < r \\ \text{for } \ell = 1, \ldots, m, & \text{if } r \leq i < p \end{cases}$$

(4) *All leaves w_i, $i = 0, \ldots, \min\{p, k\} - 1$, have period length p.*
(5) *If the text is periodic, then w_0 is the whole text, $p = |w_0| - |v_0|$ is the period length, and all implicit nodes whose length is at least p are external. Specifically, $k > |w| - 2p$ and $|v_i| < 2p$, $i = \max\{0, k-p\}, \ldots, k-1$.*

Proof. Let $w'_0 = w_0$ and let w'_i be the suffix of w'_{i-1} obtained after chopping off the first symbol of w'_{i-1}, i.e., following w'_{i-1}'s suffix link. Let $\kappa = \min\{p, k\}$. By Lemma 2.1, since $p = |w_0| - |v_0|$, the external implicit nodes are $v_i = w'_{i+p}$, $0 \leq i \leq k-1$, with $|v_i| = |v_0| - i$. We claim that $v_i = w'_{i+p}$ is a prefix of w'_i, $0 \leq i \leq k-1$. This follows immediately from the facts that (i) by definition v_0 is an ancestor (and thus a prefix) of $w'_0 = w_0$, (ii) by definition w'_i is obtained after chopping off the first symbol of w'_{i-1}, and (iii) as a consequence of Lemma 2.1, v_i also obtained after chopping off the first symbol of v_{i-1}. Since v_i is an ancestor of w'_i, the longest $w_i = w'_i$, $0 \leq i \leq \kappa - 1$, are all leaves and $|w_i| = |w_0| - i$. The leaf w_i at the edge containing v_i, $p \leq i \leq k-1$, must be the same as that at the edge containing $v_{i-p} = w'_i$. Thus $w_i = w_{i-p}$, $p \leq i \leq k-1$, establishing (1) and (2). Observe that the implicit nodes on the leaf edge to w_i are v_{i+jp}, for $0 \leq i \leq p-1$ and $0 \leq i + jp \leq k-1$. Recalling that $k = r + pm$, we get $0 \leq j \leq m$ when $0 \leq i < r$ and $0 \leq j < m$ when $r \leq i < p$, proving (3) since v_{i+jp} has length $|v_0| - i - jp = |w_0| - p - i - jp = |w_i| - (j+1)p$. We next turn to (4). Since v_0 is an ancestor of w_0 both in the suffix tree and in the suffix link trie, by Lemma 4.2, $p = |w_0| - |v_0|$ is a period length of w_0, and by the maximal length of v_0, p is the smallest such period length. The same holds for any other leaf w_i and its longest border v_i, $i = 1, \ldots, \kappa - 1$.

We finally turn to (5). Let u be an implicit node and let z_0 be the longest leaf descendant of u. We first prove that if $z_0 = u_0 u$ has period u_0, $|u_0| \leq |u|$, then u must be external. Assume it is not: then u is also ancestor of another shorter leaf $z_1 = u_1 u$ with period u_1, such that $|u_1| < |u_0|$. But z_1 is a suffix of z_0 with periods of length $|u_0|$ and $|u_1|$, such that $|u_0| + |u_1| \leq |u| + |u_1| \leq |z_1|$, and therefore by Theorem 4.1, z_1 must have a period of length $\gcd(|u_0|, |u_1|)$, and z_0 also must have period length $\gcd(|u_0|, |u_1|) < |u_0|$, clearly a contradiction. Let x be the period of the text, $w = xv$. Then by Lemma 4.2, v is an implicit node that is ancestor of w and since the text is periodic, $|x| \leq |v|$. Therefore, $v_0 = v$ must be an external implicit node, $w_0 = w$ its leaf, and $p = |w_0| - |v_0|$ the text period length. Let v_i be the suffix of v_0 of length $|v_i| = |v_0| - i$, and let w'_i be the longest leaf descendant of v_i. Since the period length of w'_i is at most p, if $|v_i| \geq p$ then v_i must be an external implicit node and $w_i = w'_i$ its leaf. This holds for all v_i such that $|v_i| = |w| - p - i \geq p$, i.e., for $i = 0, \ldots, |w| - 2p$. Thus, it must be that $k > |w| - 2p$, and the last p implicit nodes in the sequence are such that $|v_i| < 2p$, $i = \max\{0, k-p\}, \ldots, k-1$. □

We remark that irregular long periods (small overlap borders) are permitted by Theorem 4.1. However, by Theorems 4.3 and 4.4, these irregular periods must be separated from each other and from the arithmetic progression of large overlap periods by branching suffix tree nodes. Multiple implicit floating nodes within an external edge also indicate the eventual formation of new maximal repetitions in the sense of [23,27]. Theorem 4.4 also shows that an implicit suffix tree that is enhanced by the external implicit nodes, which are easy to represent, provides all suffix nodes except possibly for those in the last period of the text, missing fewer suffixes than half of the text's length. An algorithm may choose to represent the external implicit text suffixes by an arithmetic progression, or to start pro-actively inserting such implicit nodes into the tree anticipating their suffix tree locations when they eventually branch out.

5. Maintaining implicit nodes efficiently

We push Ukkonen's approach one step further and allow implicit nodes to "float" within suffix tree edges. Since implicit nodes always represent suffixes of the text, implicit nodes that do not branch out of the suffix tree as the text grows can be envisioned to be floating along the tree edges: such implicit nodes need to be updated only when they move from one edge

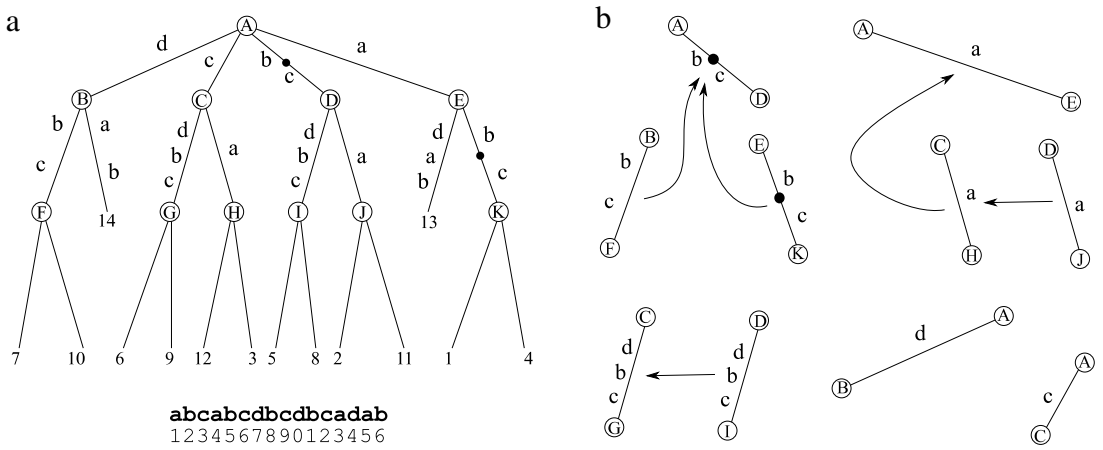

Fig. 2. (a) The suffix tree for "*abcabcdbcdbcadab*". For brevity, the last characters of suffixes are omitted and the leaves are identified by numbers giving the start position of the corresponding suffix. (b) Band trees of the internal edges in (a).

to the next. This concept is analogous to having "bounded ended edges" leading to branching nodes as opposed to "open ended" edges leading to leaves, and updates are required to skip to the next edge only when the edge boundary is reached at branching suffix tree nodes. Unfortunately, even with this proposed modification that eliminates the need to continuously maintain implicit nodes inside a suffix tree edge, the number of updates might still be too large, as shown by the example $a^n ba^n c$, which requires $\Theta(n^2)$ updates.

To reduce the number of required updates, we partition the suffix tree edges into *bands* and maintain only *one representative* for the implicit nodes within each band. We define bands as follows. Consider a suffix tree edge $e = (au, auv)$ from the node au to the node auv: the suffix link to u must be an ancestor of the suffix link to uv. If $e' = (u, uv)$ is an edge in the suffix tree, we say that the suffix tree edge e is in the same *band* as the suffix tree edge e' and connect e to e' by an edge in the *band forest*, partitioning all the edges of the suffix tree into a collection of band trees (see Fig. 2). Note that suffix tree edges correspond to band tree nodes. In the following, we use interchangeably the terms band and band tree, since there is no danger of ambiguity.

The root of a band tree is referred to as *band representative*. In our approach, we only maintain implicit nodes within band representatives, and we call them *implicit node representatives*. Since the suffix tree nodes above (respectively below) all suffix tree edges in the same band are connected by a suffix link path, an implicit node enters and exits the band together with its implicit node representative, allowing us to update only the band representatives instead of updating all the implicit nodes in the band. This definition leads to reduce the number of implicit node representative updates to $O(n)$, as shown by Theorem 5.1. Before embarking on the proof of the theorem, we need few definitions.

A band containing implicit nodes is called *active*, and it is called *inactive* otherwise. As we are interested in maintaining implicit nodes, we will focus our attention on active bands only and ignore completely inactive bands. We claim that no active band can contain simultaneously internal and external suffix tree edges. Indeed by Lemma 2.1 an external and an internal edge could belong to the same band only if the suffix chain has no implicit nodes within external or internal edges, which in turn implies that no band can be active. We can thus partition active bands into *internal* bands (containing only internal suffix tree edges) and *external* bands (containing only external suffix tree edges). We observe that it is easy to maintain information about implicit nodes floating in external edges, since an implicit node within an open ended external edge will never reach the end of that edge (as both the implicit node and the leaf at the end of the edge keep getting extended to the current end of the text). Thus, external implicit nodes will remain such until they branch out to become suffix tree leaves, and all external implicit nodes can be easily found with the arithmetic progression of Theorem 4.4. Hence, in the following we will not consider active external bands any further, and will only show how to maintain active internal bands throughout the execution of Ukkonen's algorithm.

In each active band tree, implicit nodes are distributed along a path that ends at the tree root (the band representative): this path is called the *active path* of the band. Note that an active internal band may have at most one implicit node in each of its suffix tree edges. At any time, an active band is characterized by its *start position* and *end position* in the text, which define the text interval where that band is active. In Fig. 2, the band whose representative has endpoints A and D is active, and its active path goes from the tree leaf (E, K) to the band representative (A, D). Its start position is 16, i.e., it points to the last symbol of the text "*abcabcdbcdbcadab*", and its end position is undefined yet. If the next input symbol is c, then the implicit node will reach the endpoint D of the band representative: after that, the band will no longer be active and thus its end position will be 17, i.e., the band will be active in the interval [16, 17] of the text "*abcabcdbcdbcadabc*". If the next input symbol is different from c, then the implicit node will branch out by splitting the band representative (and the corresponding band tree): the band will no longer be active and its end position will be 16, i.e., the text interval when this band representative is active will be [16, 16], corresponding to the last symbol of the text "*abcabcdbcdbcadab*".

Theorem 5.1. *An on-line algorithm has to update band representatives at most* $2n$ *times.*

Proof. We give an implementation of Ukkonen's algorithm that maintains implicit node representatives, i.e., implicit nodes in band representatives. To accomplish this task, throughout the algorithm execution we maintain a stack containing the band representatives of the internal bands that are currently active. We observe that the text intervals of all band representatives on the stack must be nested, since the endpoints of band representatives are suffix tree nodes that have suffix link paths all the way up to the suffix tree root.

At each step of Ukkonen's algorithm, we first pop from the stack all the bands whose end is reached, keeping the active node that is the end of the last popped edge, or the suffix tree root if no bands were popped from the stack (this active node is the first explicit node on the suffix chain after the active point in Ukkonen's algorithm, whereas all the longer implicit nodes are in the middle of suffix tree edges). We then check whether the current input symbol does not branch out of the suffix tree at the deepest band (the active point in Ukkonen's algorithm). If it does, then the stack is reset, new suffix tree nodes are inserted and the band tree structure is updated accordingly. In either eventuality, we repeatedly take the suffix tree edge that starts at the active node and the current input symbol and find the end of its band, the edge at the root of its band tree. That band representative is pushed onto the stack and the active node is set to the next band, the suffix link of the node at the beginning of the band tree root, repeatedly, stopping only after the suffix tree root.

We now bound the total number of band representatives that can be popped from the stack throughout the algorithm execution. We observe that any two such distinct band representatives must have either a different start or a different end position: indeed, if two band representatives share both the start and the end position, they must belong to the same band and therefore be the same. In other words, all text intervals of band representatives that have been on the stack are different. Since, as seen before, all text intervals are also nested, at most $2n$ bands can be popped from the stack while processing a string of length n. □

Theorem 5.1 can be used in an on-line implementation of Ukkonen's algorithm to maintain the band representatives and to produce implicit nodes upon queries.

Theorem 5.2. *An on-line algorithm can maintain band representatives using auxiliary data structures in* $O(n)$ *amortized time. Queries returning the implicit nodes within a specified suffix tree edge take worst-case* $O(1)$ *time.*

Proof. The algorithm makes use of two auxiliary dynamic data structures. The first data structure maintains the active band trees under operations that insert band tree leaves or delete band tree roots (splitting up the tree) in $O(1)$ amortized time each, supporting queries that return the band tree root, which is the top suffix tree edge in the band tree, in worst-case $O(1)$ time. This can be achieved with the help of data structures for dynamic nearest marked ancestors in trees [1,39]. The second data structure maintains the active path, supporting queries that check if a specific suffix tree edge that is represented by a band tree node lies in the current active path, with tree updates and queries taking worst-case $O(1)$ time. The query can be accomplished by maintaining information on the first (lowest) active band edge in the band tree and by testing whether the queried edge is on the band tree path from this first band edge and the band tree root. This can be implemented efficiently with data structures that check for ancestor relationship [4,15,36].

The stack in Theorem 5.1 is then implemented using these data structures, where each band that is pushed on the stack has its band representative, the band tree root, updated and the active band path set, leading to an $O(n)$ on-line implementation of Ukkonen's suffix tree algorithm. Insertions in each batch are effectuated from shallow to deep, from the shortest inserted suffix towards the longest (reverse of Ukkonen's algorithm using reverse suffix links), so that only band tree roots are deleted at each step and the split edge's parts are inserted as leaves into their appropriate band trees. Queries asking for the implicit nodes contained in some specified suffix tree edge can be implemented by first locating the root of the corresponding band tree, and then by checking if the queried edge lies in the active path of the band tree. If the edge lies in the active path, the band tree root gives the single implicit node in the band representative for internal suffix tree bands (Theorem 4.3). As mentioned previously, all implicit nodes in external suffix tree bands can be returned by means of the arithmetic progression of Theorem 4.4. □

6. Conclusions

We proved some new combinatorial properties about suffix trees that appear to be mostly of theoretical interest; we are curious whether they can be used in new applications. For example, the intermediate suffix trees could be used as an on-line index to report all occurrences of a pattern in the text in time proportional to the pattern length and the number of reported occurrences. However, one could also use instead the suffix tree of the reverse text [38] and the closely related *directed acyclic word graph (DAWG)* [6,7,9,11,13], both of which undergo only $O(n)$ structural changes while being updated on-line.

There exist off-line linear-time algorithms for suffix tree and suffix array construction over larger integer alphabets [17,18,24–26,35] (DAWG and Aho-Corasick Automata by reductions [8,16]). We are curious whether linear-time on-line suffix tree algorithms exist over large integer alphabets.

Acknowledgments

We thank the anonymous referees for their remarks, and Maxime Crochemore, Roberto Grossi, Gadi Landau and Filippo Mignosi for several discussions and comments about this work.

The first author was partially supported by the European Research Council (ERC) Project SFEROT and by the Israeli Science Foundation Grants 35/05, 686/07, 347/09 and 864/11. The second author was partially supported by the 7th Framework Programme of the EU (Network of Excellence "EuroNF: Anticipating the Network of the Future — From Theory to Design") and by MIUR, the Italian Ministry of Education, University and Research, under Project AlgoDEEP.

References

[1] A. Amir, M. Farach, R. Idury, J.L. Poutré, A. Schäffer, Improved dynamic dictionary-matching, Inform. and Comput. 119 (1995) 258–282.
[2] A. Apostolico, the myriad virtues of subword trees, in: A. Apostolico, Z. Galil (Eds.), Combinatorial Algorithms on Words, in: NATO ASI Series F, vol. 12, Springer-Verlag, Berlin, Germany, 1985, pp. 85–96.
[3] A. Apostolico, F. Preparata, Optimal off-line detection of repetitions in a string, Theoret. Comput. Sci. 22 (1983) 297–315.
[4] M.A. Bender, R. Cole, E.D. Demaine, M. Farach-Colton, J. Zito, Two simplified algorithms for maintaining order in a list, in: R.H. Möhring, R. Raman (Eds.), ESA, in: Lecture Notes in Computer Science, vol. 2461, Springer, 2002, pp. 152–164.
[5] J. Berstel, Transductions and Context-Free Languages, Teubner-Verlag, 1979, Revised version is available electronically as http://www-igm.univ-mlv.fr/~berstel/LivreTransductions/LivreTransductions14dec2009.pdf.
[6] A. Blumer, J. Blumer, D. Haussler, A. Ehrenfeucht, M. Chen, J. Seiferas, The smallest automaton recognizing the subwords of a text, Theoret. Comput. Sci. 40 (1985) 31–55.
[7] A. Blumer, J. Blumer, D. Haussler, R. McConnel, A. Ehrenfeucht, Complete inverted files for efficient text retrieval and analysis, J. Assoc. Comput. Mach. 34 (3) (1987) 578–595.
[8] D. Breslauer, R. Hariharan, Optimal parallel construction of minimal suffix and factor automata, Parallel Process. Lett. 6 (1) (1996) 35–44.
[9] D. Breslauer, G.F. Italiano, Near real-time suffix tree construction via the fringe marked ancestor problem, in: SPIRE, in: Lecture Notes in Computer Science, vol. 7024, Springer, 2011, pp. 156–167.
[10] D. Breslauer, G.F. Italiano, On suffix extensions in suffix trees, in: SPIRE, in: Lecture Notes in Computer Science, vol. 7024, Springer, 2011, pp. 301–312.
[11] M. Chen, J. Seiferas, Efficient and elegant subword-tree construction, in: A. Apostolico, Z. Galil (Eds.), Combinatorial Algorithms on Words, in: NATO ASI Series F, vol. 12, Springer-Verlag, Berlin, Germany, 1985, pp. 97–107.
[12] H. Cohen, E. Porat, Range non-overlapping indexing, in: Y. Dong, D.-Z. Du, O.H. Ibarra (Eds.), ISAAC, in: Lecture Notes in Computer Science, vol. 5878, Springer, 2009, pp. 1044–1053.
[13] M. Crochemore, Transducers and repetitions, Theoret. Comput. Sci. 12 (1986) 63–86.
[14] M. Crochemore, W. Rytter, Text Algorithms, Oxford University Press, 1994.
[15] P.F. Dietz, D.D. Sleator, Two algorithms for maintaining order in a list, in: STOC, ACM, 1987, pp. 365–372.
[16] S. Dori, G.M. Landau, Construction of Aho Corasick automaton in linear time for integer alphabets, Inf. Process. Lett. 98 (2) (2006) 66–72.
[17] M. Farach, Optimal suffix tree construction with large alphabets, in: FOCS, 1997, pp. 137–143.
[18] M. Farach-Colton, P. Ferragina, S. Muthukrishnan, On the sorting-complexity of suffix tree construction, J. ACM 47 (6) (2000) 987–1011.
[19] N. Fine, H. Wilf, Uniqueness theorems for periodic functions, Proc. Amer. Math. Soc. 16 (1965) 109–114.
[20] E. Fredkin, Trie memory, Commun. ACM 3 (9) (1960) 490–499.
[21] R. Giegerich, S. Kurtz, From ukkonen to mccreight and weiner: a unifying view of linear-time suffix tree construction, Algorithmica 19 (3) (1997) 331–353.
[22] D. Gusfield, Algorithms on Strings, Trees, and Sequences - Computer Science and Computational Biology, Cambridge University Press, 1997.
[23] D. Gusfield, J. Stoye, Linear time algorithms for finding and representing all the tandem repeats in a string, J. Comput. Syst. Sci. 69 (4) (2004) 525–546.
[24] J. Kärkkäinen, P. Sanders, S. Burkhardt, Linear work suffix array construction, J. ACM 53 (6) (2006) 918–936.
[25] D.K. Kim, J.S. Sim, H. Park, K. Park, Constructing suffix arrays in linear time, J. Discrete Algorithms 3 (2–4) (2005) 126–142.
[26] P. Ko, S. Aluru, Space efficient linear time construction of suffix arrays, J. Discrete Algorithms 3 (2–4) (2005) 143–156.
[27] R.M. Kolpakov, G. Kucherov, Finding maximal repetitions in a word in linear time, in: FOCS, 1999, pp. 596–604.
[28] S. Kosaraju, Computation of squares in a string, in: Proc. 5th Symp. on Combinatorial Pattern Matching, in: Lecture Notes in Computer Science, vol. 807, Springer-Verlag, Berlin, Germany, 1994, pp. 146–150.
[29] M. Lothaire, Algebraic Combinatorics on Words, Cambridge University Press, 2005.
[30] U. Manber, E.W. Myers, Suffix arrays: A new method for on-line string searches, SIAM J. Comput. 22 (5) (1993) 935–948.
[31] E. McCreight, A space economical suffix tree construction algorithm, J. Assoc. Comput. Mach. 23 (1976) 262–272.
[32] D.R. Morrison, PATRICIA — practical algorithm to retrieve information coded in alphanumeric, J. ACM 15 (4) (1968) 514–534.
[33] M. Salson, T. Lecroq, M. Léonard, L. Mouchard, A four-stage algorithm for updating a Burrows–Wheeler transform, Theoret. Comput. Sci. 410 (43) (2009) 4350–4359.
[34] M. Salson, T. Lecroq, M. Léonard, L. Mouchard, Dynamic extended suffix arrays, J. Discrete Algorithms 8 (2) (2010) 241–257.
[35] T. Shibuya, Constructing the suffix tree of a tree with a large alphabet, in: A. Aggarwal, C.P. Rangan (Eds.), ISAAC, in: Lecture Notes in Computer Science, vol. 1741, Springer, 1999, pp. 225–236.
[36] A.K. Tsakalidis, Maintaining Order in a Generalized Linked List, Acta Inf. 21 (1984) 101–112.
[37] E. Ukkonen, On-line construction of suffix trees, Algorithmica 14 (3) (1995) 249–260.
[38] P. Weiner, Linear pattern matching algorithms, in: Proc. 14th Symposium on Switching and Automata Theory, 1973, pp. 1–11.
[39] J. Westbrook, Fast incremental planarity testing, in: Proc. 19th International Colloquium on Automata, Languages, and Programming, in: Lecture Notes in Computer Science, vol. 623, Springer-Verlag, Berlin, Germany, 1992, pp. 342–353.